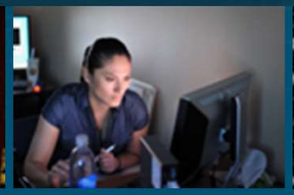
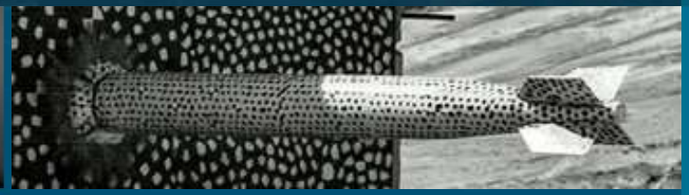
Sandia National Laboratories
Privacy Enhancing Technologies Working Group

Two Novel Approaches to Secure Computing with Both Functional and Data Privacy

Wednesday, May 11th, 2022

Michael P. Frank, Center for Computing Research

with Ryan Kao, Nick Pattengale, Vlad Kolesnikov, et al.

Approved for Public Release, SAND2022-7575 PE

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline of Talk

A few relevant definitions:

- Secure computing paradigms
- Requirements pertaining to functional & data privacy
- Security models

Brief review of some of the existing, more "mainstream" secure computing paradigms:

- (Secure) Multiparty Computation (MPC)
- Fully Homomorphic Encryption (FHE)
- Indistinguishability Obfuscation (IO)

The focus of today's talk: Overview:

1. Blockchain-based secure computation using garbled functions (a.k.a. GABLE project)
   - Focus of an LDRD at Sandia "Blockchain Derived Secure Computation" that ran from 2017-2020.
2. Efficient block ciphers based on reversible computing, & a new derived secure computing paradigm
   - Work out of a group at Boston University (BU) and the University of Central Florida (UCF).

What do we mean by "Secure Computing?"

Computation featuring (strong) security properties.

The secure computing concept typically includes a requirement for ***privacy***, which may include:

- ***Data privacy*** – Plaintext data may not (feasibly) be inferred by any unauthorized parties.
  - In general, "unauthorized parties" could even include the entity that is running the computation!
- ***Functional privacy*** – The nature of the function being computed may not (feasibly) be inferred by unauthorized parties.
  - Again, most generally, unauthorized parties could include the entity that is running the computation.

Why might we want these properties? Some examples:

- ***Data privacy*** – For operations combining confidential data from different privacy/jurisdictional domains; also allows outsourcing of computational services without compromising confidentiality of data.
- ***Functional privacy*** – Can obscure proprietary algorithmic innovations, or protect strategic intent; can protect *e.g.* restricted cryptographic capabilities (make them available for use by unauthorized parties).

In addition to privacy properties, we may also want other security-related properties, such as *robustness, reliability, auditability/verifiability, non-repudiability, integrity*…

- We'll not go into these properties in great detail in this talk, but will mention some of them in passing.

A little more detail on privacy requirements:

In general, we may wish for the privacy of one or more of the following to be protected…

- Input data to a computation
- Intermediate data internal to the computation
- Information about the structure/function/implementation of the computation itself
- Output data (results) from a computation

And in general, we may wish for data to be protected from access by one or more of:

- (Other) parties providing input to the same computation
- Entities that are executing the computation
- (Other) parties that are receiving output from the same computation
- Other parties not involved in the computation who may happen to intercept (encrypted) data

Further, in general one could specify more fine-grained security profiles with various levels of granularity, specifying which authorities are authorized to access which specific pieces of data.

- An "authority" is in general an abstract role which may be held by one or more parties.

Similarly, data and functional ***integrity*** denotes that it is infeasible for unauthorized parties to alter/write invalid data or functional behavior in ways that are undetectable to honest participants.

- Like as with read permissions, write permissions may similarly be fine-grained.

Security Model

It's of course important when analyzing the capabilities of any secure computing paradigm, or a more detailed protocol, to pay careful attention to the ***security model***, which addresses things like:

- What kind of security properties are we even trying to provide?
- What assumptions do we make regarding the behavior/trustworthiness of authorized participants in a given protocol? What actions are they allowed/not allowed to do, to be considered in compliance w. the protocol?
  - And, what security properties, if any, still remain satisfied even in the presence of particular types of non-compliance?
- What assumptions do we make regarding the kinds of capabilities that an adversary may or may not have?
  - How much unauthorized information, and what specific information, are we assuming an adversary may potentially possess?
  - How many/which participants in the protocol do we assume may potentially be (partially or fully) compromised by an adversary?
  - At what point(s) in the protocol sequence diagram(s) may an adversary potentially insert/delete/modify/delay messages?
  - What kind of computational capabilities do we assume an adversary may or may not have?

The differences in security models between different secure computing paradigms are of course important to keep in mind when comparing different paradigms (and detailed implementations).

- *E.g.*, it isn't always possible to substantiate that one approach is overall strictly "better" or "worse" than another *in general* if their security models are making very different, and non-comparable assumptions.
- Whether a given paradigm/protocol is appropriate to a given real-world scenario or not will typically depend *a lot* on how appropriate the assumptions made in the security model are to the real-world circumstances.

The two new secure computing approaches we'll discuss have somewhat different security models.

Brief Overview of Some Existing Secure Computing Paradigms

(Secure) Multiparty Computation (MPC)

- Typical setup: Several parties want to jointly compute a function of their private data, but *without revealing any more information to each other about their private data than is necessarily implied by the (shared) result of the computation.*
  - This setup could be further refined, by, *e.g.*, not sharing *all* of the output data with *all* of the participants, *i.e.*, fine-grain the reader authority.
    - The requirement then becomes, no more input information is leaked to any party than what's implied by the outputs they are authorized to read.
- Standard MPC *does not care about functional privacy*, but, it can be *extended* to also support functional privacy using the concept of a *universal circuit*, so that the function to be executed just becomes another private input.

Fully Homomorphic Encryption (FHE)

- Setup: *Compute directly on encrypted data, without decryping it.* Given a function F, want $F'(E(d)) = E(F(d))$.
- Key breakthrough: Gentry '09, "bootstrapped" method using lattice crypto, doi:10.1145/1536414.1536440.
  - Works, but typically has very large overhead.
- More efficient techniques have been developed since then, but still have significant complexity.

Indistinguishability Obfuscation (IO)

- ***Obfuscation*** attempts to replace a program with an unintelligible, but functionally equivalent program.
- Barak *et al.* (2001, 2012) showed that *perfect* (*i.e.*, black-box) obfuscation is impossible!
  - Defined ***indistinguishability obfuscation*** as a weaker notion: Can't tell *which* of the functionally equivalent programs was obfuscated.
- Goldwasser & Rothblum (2007) defined a closely related notion of ***Best-Possible Obfuscation***:
  - Definition: Obfuscated program leaks *no more* information than *any other* functionally equivalent program.
  - This definition is equivalent to IO for the case of *efficient* obfuscation (which is the most interesting case for practical applications).

Two New Approaches We'll Discuss

(#1) The approach that was investigated in the "blockchain-derived secure computing" LDRD:

- This can be considered an application/variation on the basic concepts of garbled universal circuits (GUC).
- Security model is slightly different from the usual setup, implementations explored were slightly different.
- Main application:
  - Apply techniques of garbled computation to demonstrate a capability for secure computing on *public blockchains* (such as Ethereum).
  - We built and demonstrated a detailed approach called "Garbled Autonomous Bots Leveraging Ethereum" (GABLE).
- Can also be viewed as providing a somewhat *more limited version* of some of the capabilities that you might want out of FHE (obscuration of user data) and/or IO (obscuration of detailed functionality).
  - We'll discuss this comparison briefly.

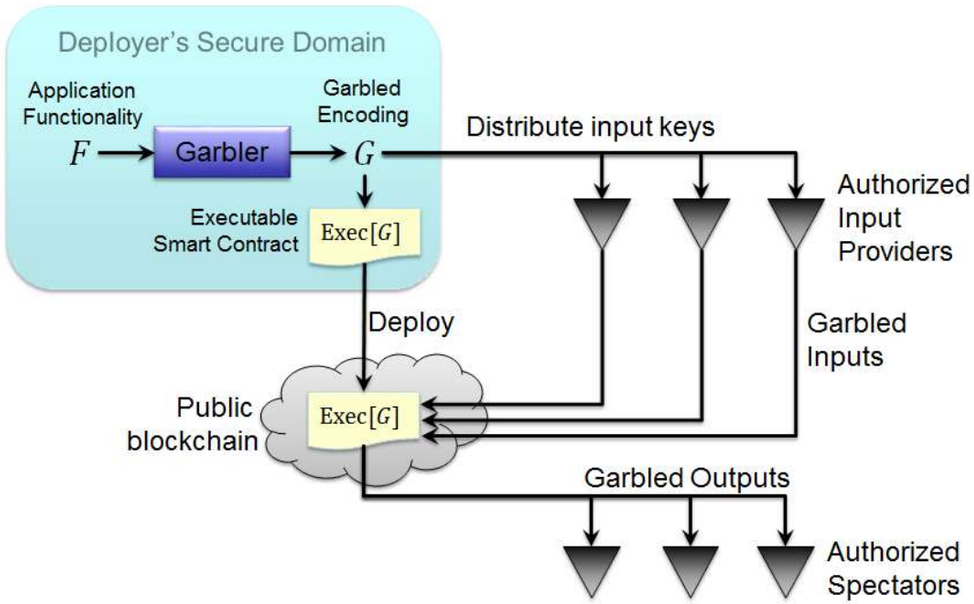
(#2) The approach from the BU/UCF group (Chamon, Jakes-Schauer, Mucciolo, & Ruckenstein):

- This one is billed primarily as an *alternative to homomorphic encryption* (*i.e.*, serving many of the same purposes).
  - But in suitable contexts, as with GABLE, it can also apparently provide a limited version of some of the same characteristics you might want out of indistinguishability obfuscation / best possible obfuscation.
- Includes a technique for constructing efficient (logarithmic-depth) block ciphers, apparently highly secure.
  - Security argument is based on concepts of chaos from statistical physics – somewhat unconventional formal approach.
- They build on this idea to construct an FHE-like approach for operating directly on encrypted data.
  - The *specific* key is used to *derive* a program that can operate on encrypted data, but the key/plaintext data can't be extracted from the program!
    - Like FHE, effectively separates the security domain of the input provider (who knows the data & key) from that of the entity *executing* the program (who doesn't).
    - **As far as I am aware, this is a somewhat novel/unique, but potentially quite useful setup for secure computing!**
- Construction of the secure computation is reasonably efficient (polynomial overhead).
- The authors are looking to get more feedback from professional cryptographers, esp. FHE experts

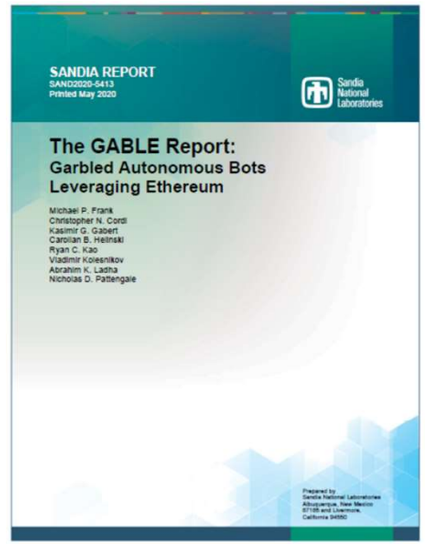
Blockchain-Derived Secure Computing LDRD / GABLE System

Basic setup:

- Some entity C ("the *Company*") has a private function F requiring robust online execution.
- They generate G, a *garbled* (encrypted) version of F, with associated *input keys*.
  - Input keys are distributed to respective *input providers*, and an executor for G is published (*e.g.* on the blockchain)



- The executor accepts inputs from input providers, and executes the garbled program (once).
  - Or, one variant (not yet published) executes the program off-chain and only verifies results on-chain, for greater efficiency.
  - Potentially could also be extended to a multiple-use scenario using ***reusable garbled circuits***. (Needs further study.)
- Authorized readers/spectators may retrieve/interpret results (also using respective keys).



SAND2020-5413

Cryptology ePrint Archive: Report 2022/398

Auditable, Available and Resilient Private Computation on the Blockchain via MPC

Christopher Cordi and Michael P. Frank and Kasimir Gabert and Carollan Helinski and Ryan C. Kao and Vladimir Kolesnikov and Abrahim Ladha and Nicholas Pattengale

Abstract: Simple but mission-critical internet-based applications that require extremely high reliability, availability, and verifiability (e.g., auditability) could benefit from running on robust public programmable blockchain platforms such as Ethereum. Unfortunately, program code running on such blockchains is normally publicly viewable, rendering these platforms unsuitable for applications requiring strict privacy of application code, data, and results. In this work, we investigate using MPC techniques to protect the privacy of a blockchain computation. While our main goal is to hide both the data and the computed function itself, we also consider the standard MPC setting where the function is public. We describe GABLE (Garbled Autonomous Bots Leveraging Ethereum), a blockchain MPC architecture and system. The GABLE architecture specifies the roles and capabilities of the players. GABLE includes two approaches for implementing MPC over blockchain: Garbled Circuits (GC), evaluating universal circuits, and Garbled Finite State Automata (GFSA). We formally model and prove the security of GABLE implemented over garbling schemes, a popular abstraction of GC and GFSA from (Bellare et al, CCS 2012). We analyze in detail the performance (including Ethereum gas costs) of both approaches and discuss the trade-offs. We implement a simple prototype of GABLE and report on the implementation issues and experience.

Category / Keywords: cryptographic protocols / MPC, garbled circuits, blockchain, garbled FSA

Original Publication (with major differences): 6th International Symposium on Cyber Security, Cryptology and Machine Learning (CSCML 2022)

Date: received 27 Mar 2022

Contact author: mpfrank at sandia gov, vlad kolesnikov at gmail com

Available format(s): PDF | BibTeX Citation

Note: To be published at CSCML 2022.

Version: 20220328:144554 (All versions of this report)

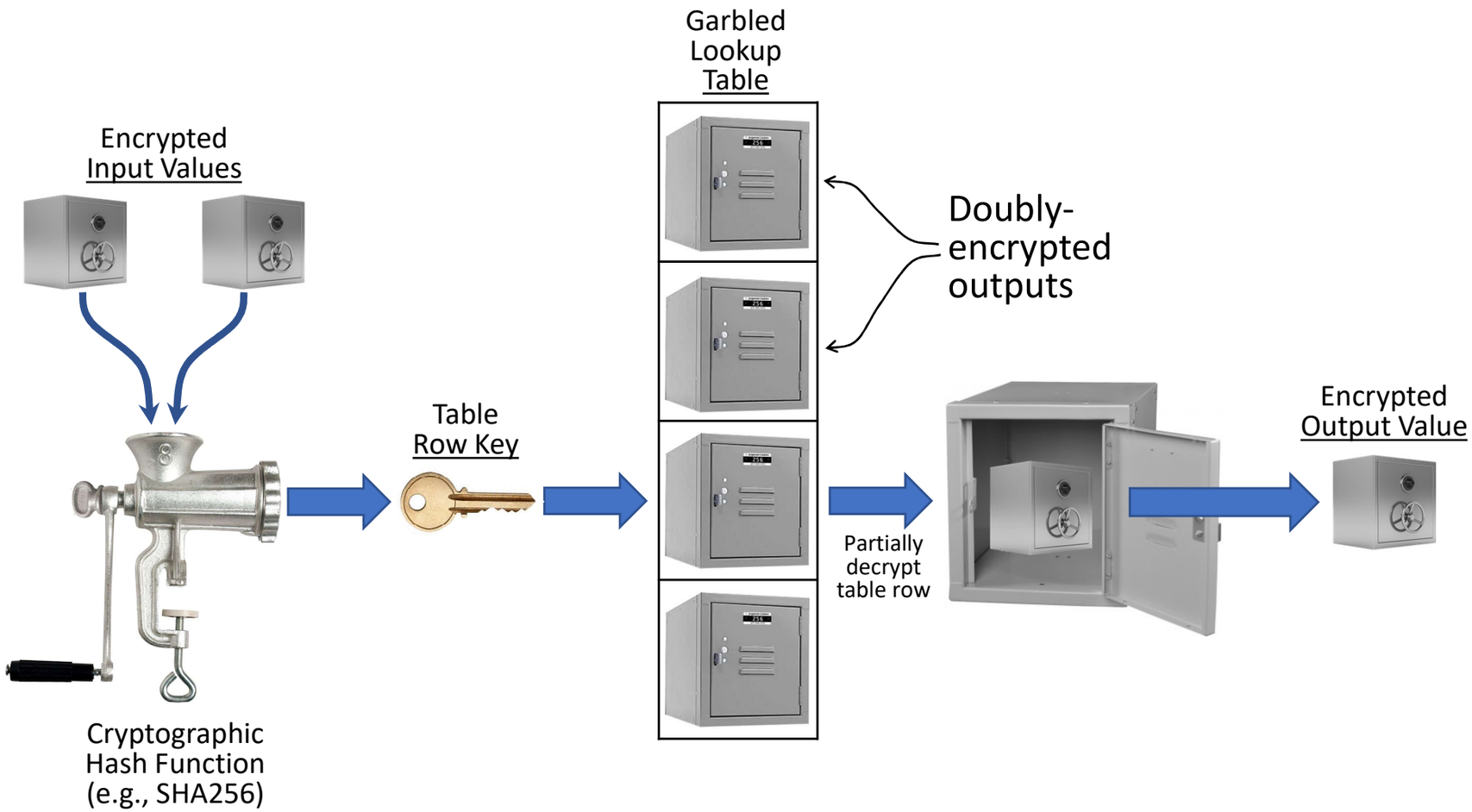
Short URL: ia.cr/2022/398

ia.cr/2022/398

Garbled Table Abstraction

An important primitive for (use-once) private function evaluation (PFE) on encrypted data.

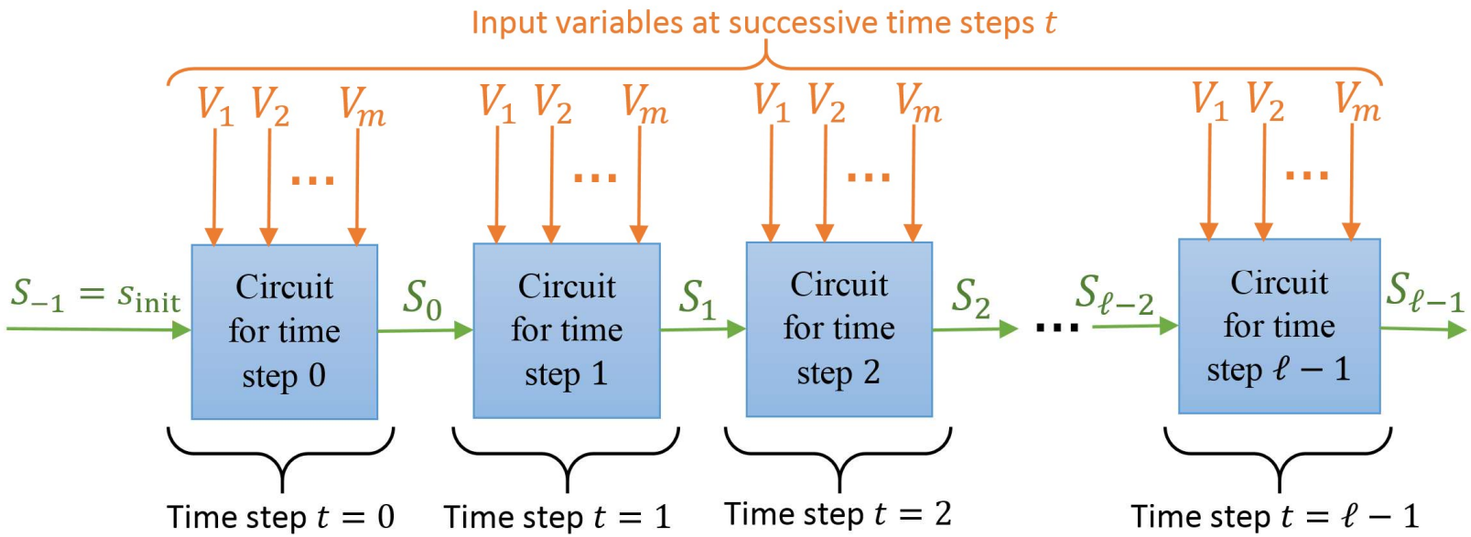
- Functions are encoded as lookup tables, but may be chained together to build up complex functionality.
  - We explored two different chaining methods: Garbled Finite State Machines (GFSM) and Garbled Configurable Universal Circuits (GCUC)



Finite-State Machine Execution Picture

Application logic is represented by a finite-state machine (FSM)

- State machine execution is *unrolled* to prevent replay attacks
  - Each step of execution is *independently* garbled

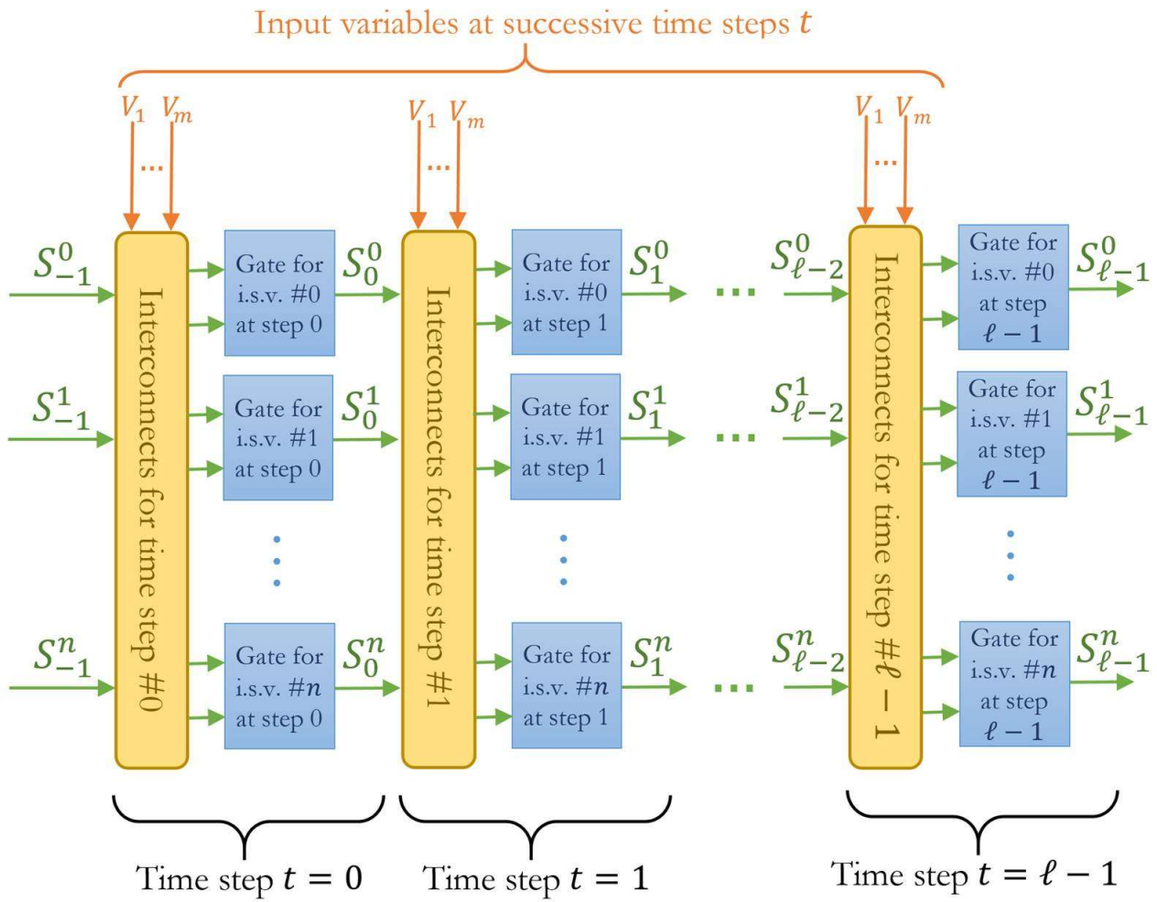


Randomly-generated keys associated with assignments of values to input variables $V_i = v_i^j$ (for each time step t) are given to authorized *input providers*

Configurable Universal Circuits

This is just a refinement of the previous picture.

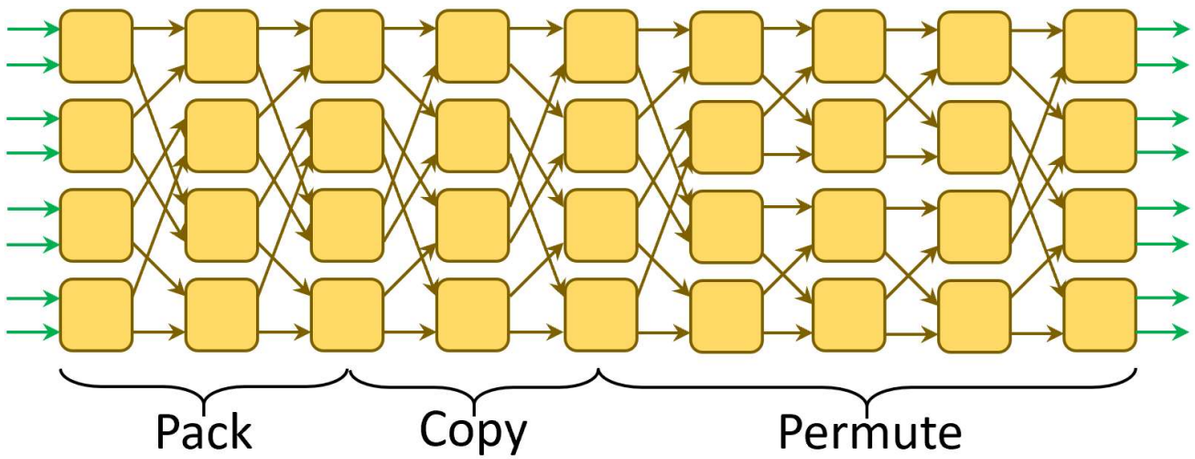
- Now, state-updating circuitry is subdivided into an *interconnect fabric* and a layer of *application gates*.



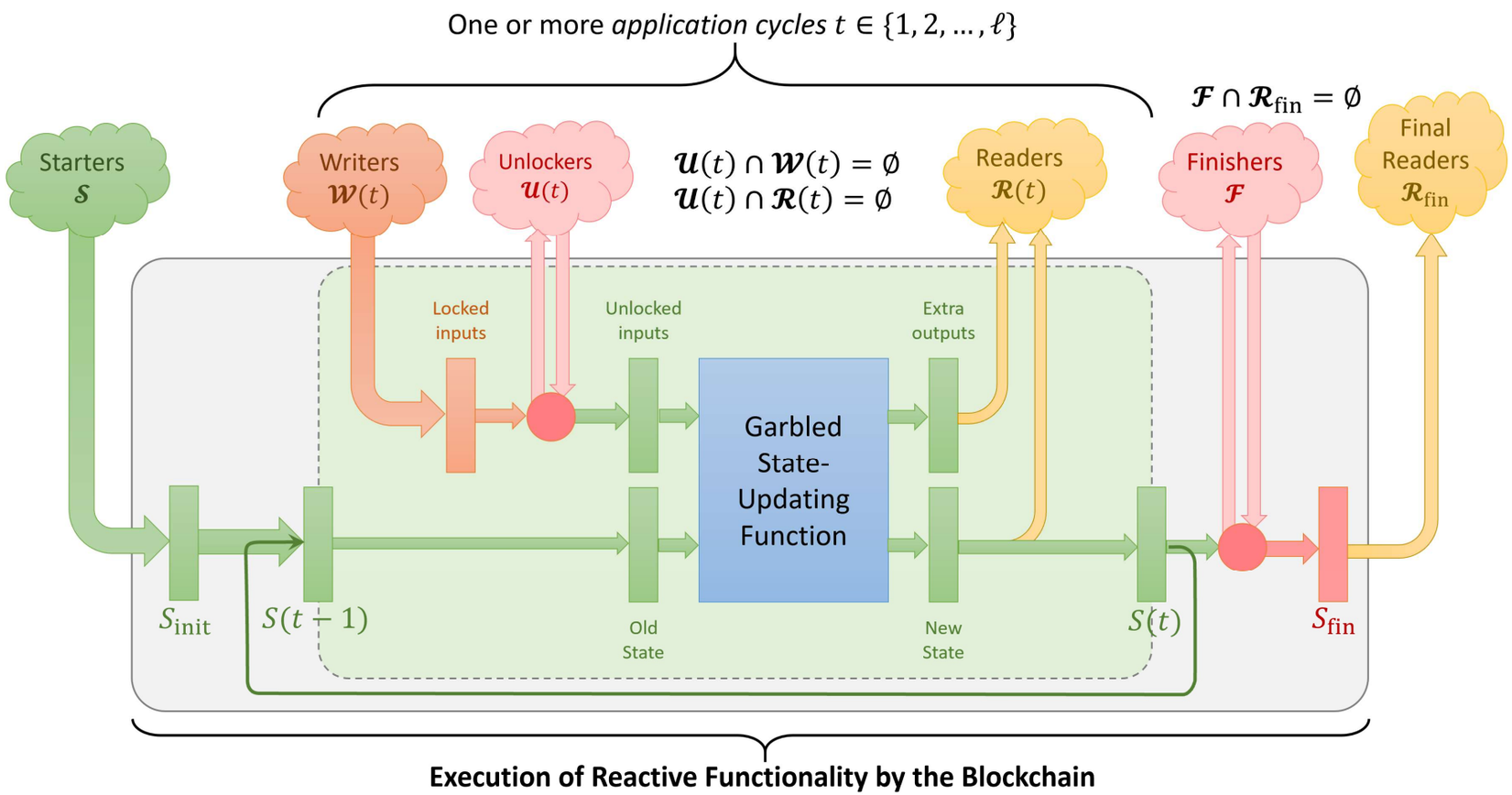
Garbling the Interconnect Fabric

Can embed an *arbitrary* interconnect fabric in a $(\log w)$-depth *Thompson network*:

- Each switch block here can be garbled similarly to a pair of Boolean gates



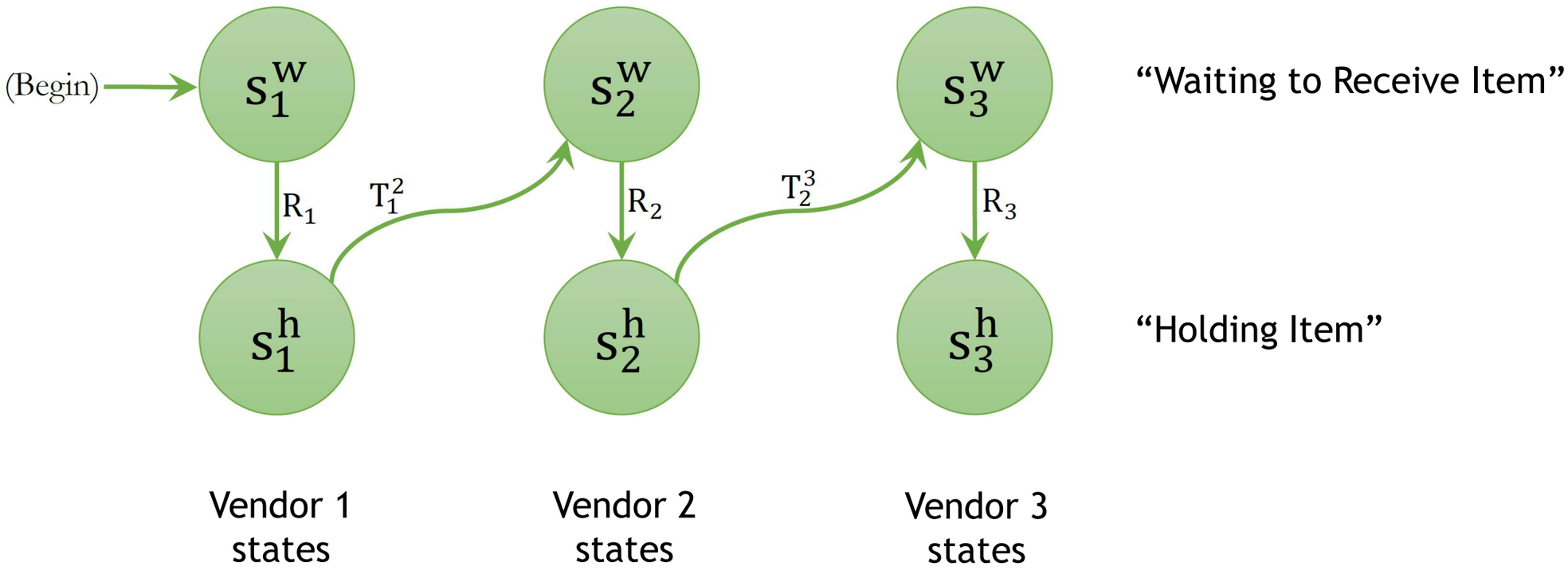
Adding Unlockers to Prevent Lookahead



GABLE Demo #1 – "Supply Chain"

"Toy" example to demonstrate garbled finite-state machine capability

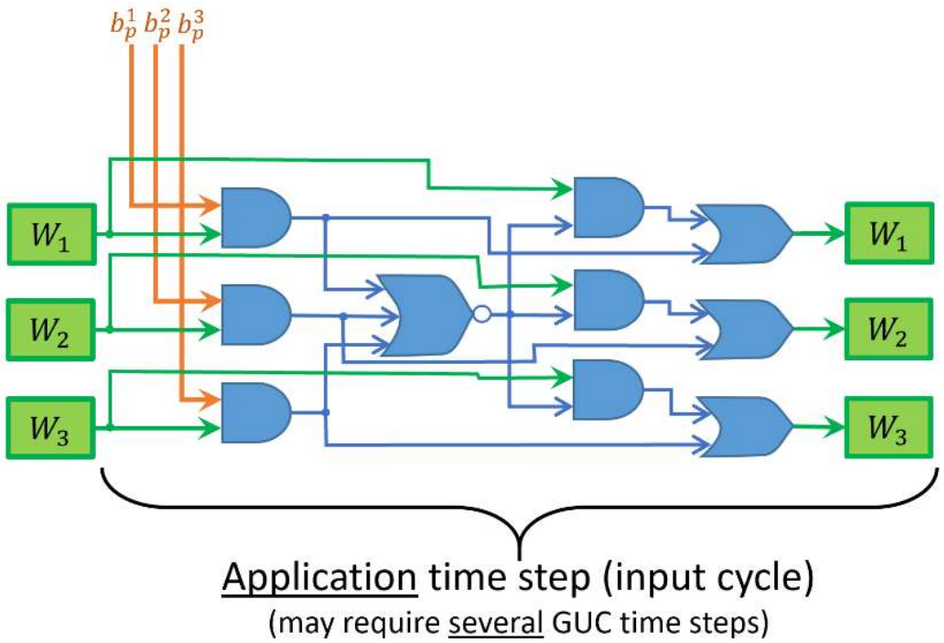
- Models a tracking system for items passing down a supply chain as different vendors do their value-adds
  - In this demo, each vendor can only "see" their own states – can't see the structure/activity on the rest of the supply chain

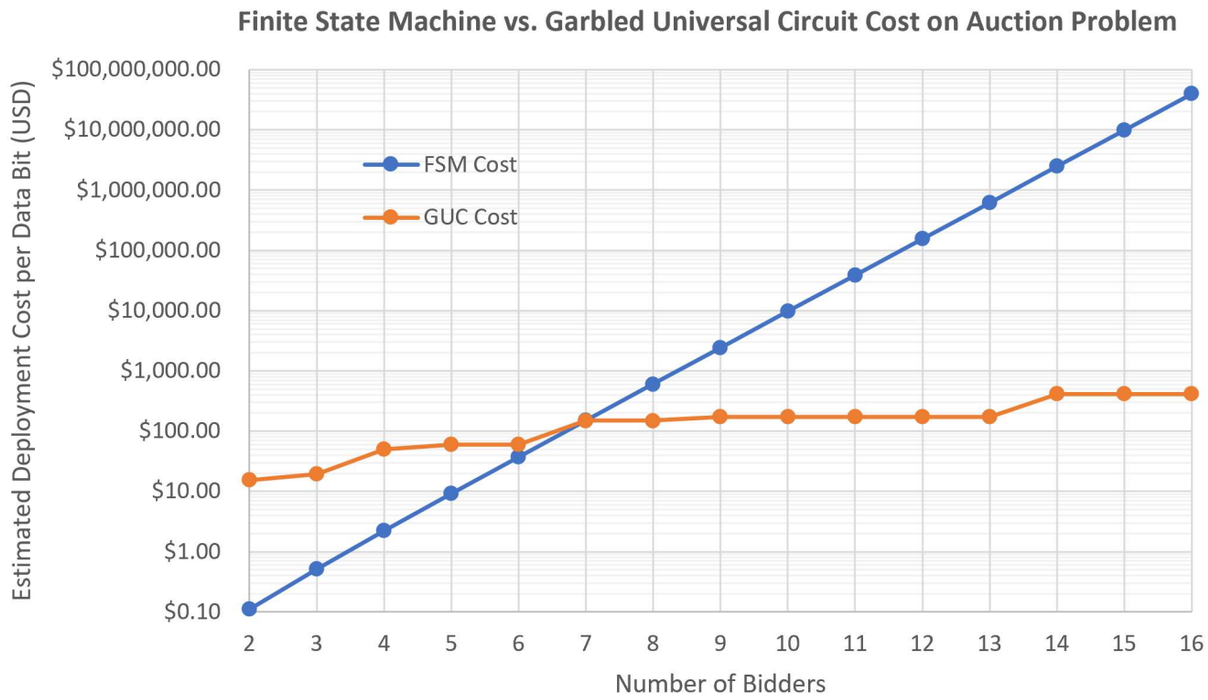


GABLE Demo #2 – "Multi-Party Auction"

A simple circuit suffices to update the state in a bit-serial multi-party auction protocol

- This test case was used to concretely compare efficiency between the GFSM and GCUC approaches
  - Logarithmic-overhead GCUC approach beats exponential-complexity GFSM approach for >7 bidders.





Chamon et al. approach

Two important papers so far:

- C. Chamon, E.R. Mucciolo, & A.E. Ruckenstein, "Quantum statistical mechanics of encryption: reaching the speed limit of classical block ciphers," arXiv:2011.06546 [cs.CR] (Nov. '20, updated Mar. '22).
- C. Chamon, J. Jakes-Schauer, E.R. Mucciolo, and A.E. Ruckenstein, "Encrypted Operator Computing: an alternative to Fully Homomorphic Encryption," arXiv:2203.08876 [cs.CR] (March 2022).

The first paper shows a construction via which one can generate apparently *optimal* block ciphers by composing only $O(n \log n)$ 2- and 3-bit classical reversible gates.

- Argument for its strength is based on thermodynamic concepts of quantum and classical chaos!

The second paper builds on the first by describing a novel secure computing scheme called ***Encrypted Operator Computing*** (EOC).

- Uses a random block cipher E, as per the first paper, to encrypt/decrypt data. (Quantum "change of basis.")
  - Can also be extended to support an asymmetric cryptography framework
- Given a function F to compute, generates an *encrypted operator* F^E that evaluates F on input data $E(x)$ without ever decrypting $E(x)$! (Computes directly "in the encrypted basis.")
  - The representation of F^E "scrambles" F and E together in such a way that neither of them can be extracted, given just F^E!
- Allows separating *knowledge of the function* and *knowledge of the data* into separate security domains.
  - Key difference from FHE is that *both* the function and the data depend on the "key" E, but you can't infer either one from the other!

Classical Reversible Logic Gates

The traditional concept of classical (*i.e.*, not specifically quantum) fully-reversible logic gates dates back to work by Ed Fredkin and Tommaso Toffoli at MIT in the late '70s / early '80s.

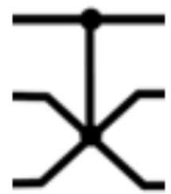
- Simply put, such operators apply a permutation or bijective map to the space of bit vectors.
- The concept was later generalized to (unitary) "quantum gates" by David Deutsch in the late '80s.
  - The whole field of (gate-based) quantum computing is based on this.

The field of *reversible computing* studies what can be done using operations like these.

- Shown to be *computation universal* (with some overheads) by Lecerf ('63) and Bennett ('73).
- There are important applications in *energy-efficient computing* (*e.g.*, beating the Landauer limit).
- In cryptography, these operations are also quite useful in symmetric cryptography as well as (perhaps counter-intuitively) in the construction of one-way functions.
  - *E.g.*, every step in SHA-256 is reversible up until you discard the W_t vector (which is derived from the input message block).

The Chamon *et al.* constructions are entirely based on reversible operations!

- Illustrates the incredible power of this paradigm…



Block Ciphers

A classic (substitution) cipher C is bijective map from the message alphabet Σ to itself, $C: \Sigma \to \Sigma$.

- *I.e.*, a permutation of the message alphabet. (Basic example: Rot-13 of the Roman alphabet.)

A block cipher B is similar, except that it operates on entire fixed-size *blocks* of data from the message, *e.g.*, 512-bit blocks. $B: \{0,1\}^n \to \{0,1\}^n$.

- An advantage of using large blocks is that repetition of any given block is much rarer, rendering brute-force cryptanalysis much more difficult.
  - An idealized case would be if B was a *random* permutation, meaning there are no regularities between encryption of different blocks.
    - However, a truly random permutation on a large blocksize would be infeasible to specify: Lookup table with 2^n entries.
- In practice, block ciphers tend to use *pseudo-random* permutations from some more limited family.
  - Some requirements:
    - There have to be too many pseudo-random permutations in the family to feasibly search them all (*e.g.*, large seed lengths, generated with high entropy).
    - The pseudo-random permutation should be infeasible to distinguish from a random permutation given a polynomial number of examples.
- It's been known for a while that good pseudo-random permutations can be obtained by randomly composing small (at least 3-bit) random permutations – *i.e.*, reversible logic gates.

This problem is closely related to the problem of ***information scrambling*** by random quantum circuits in the context of quantum computing – relevant to *quantum supremacy* demonstrations.

- Best results prior to this one demonstrated good pseudo-random quantum scrambling with $O(n \log^2 n)$ gates.
  - Example reference: A. Harrow & S. Mehraban, "Approximate unitary t-designs by short random quantum circuits using nearest-neighbor and long-range gates," arXiv:1809.06957 [quant-ph] (2018).

The Chamon *et al.* work improves this bound to $O(n \log n)$ for the classical case.

Overview of Methods in new Chamon *et al.* block cipher work

Feasibility of plaintext & ciphertext attacks related to ***out-of-time order correlators*** (OTOCs).

- A measure of quantum chaos – but also remains relevant in a classical reversible context.
- Four-point correlation function between two suitable probe operators at different times

In this work, the operators used for defining the OTOCs are called "strings"

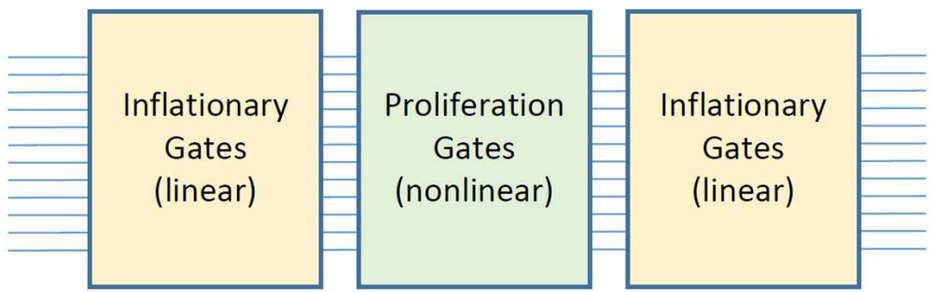
- In this paper, ***string*** is a technical term for a composition of operators describing the input bits being modified and the output bits being measured
  - Corresponds to attacks in which the attacker examines the effect on the output (ciphertext) of changing certain input (plaintext) bits

The analytical methods used in this work involve translating the string operators from the computational (bit) space into a dual ***string space*** to facilitate analysis.

- In the dual space, nonlinear reversible gates transform individual strings into *superpositions* of strings.

A particular 3-layer architecture (argued to be optimal) is studied/analyzed in this work:

- A layer of *nonlinear* 3-bit gates, bookended by layers of *linear* gates.
- The *linear* gates (*e.g.*, CNOT) "expand" the strings (to cover more bits)
- The *nonlinear* gates (*e.g.*, Toffoli) "proliferate" the strings (turning pure strings into superpositions)
- Gates organized in hierarchical structure to exponentially spread info.



As the circuit becomes deeper, the OTOCs vanish, indicating that the resulting permutation is becoming indistinguishable from random.

- Physicists are happy with this metric – is it seen as adequate / sufficiently rigorous by cryptographers?

arXiv:2011.06546 [cs.CR]

Some Analytical/Numerical Results on the new Block Cipher

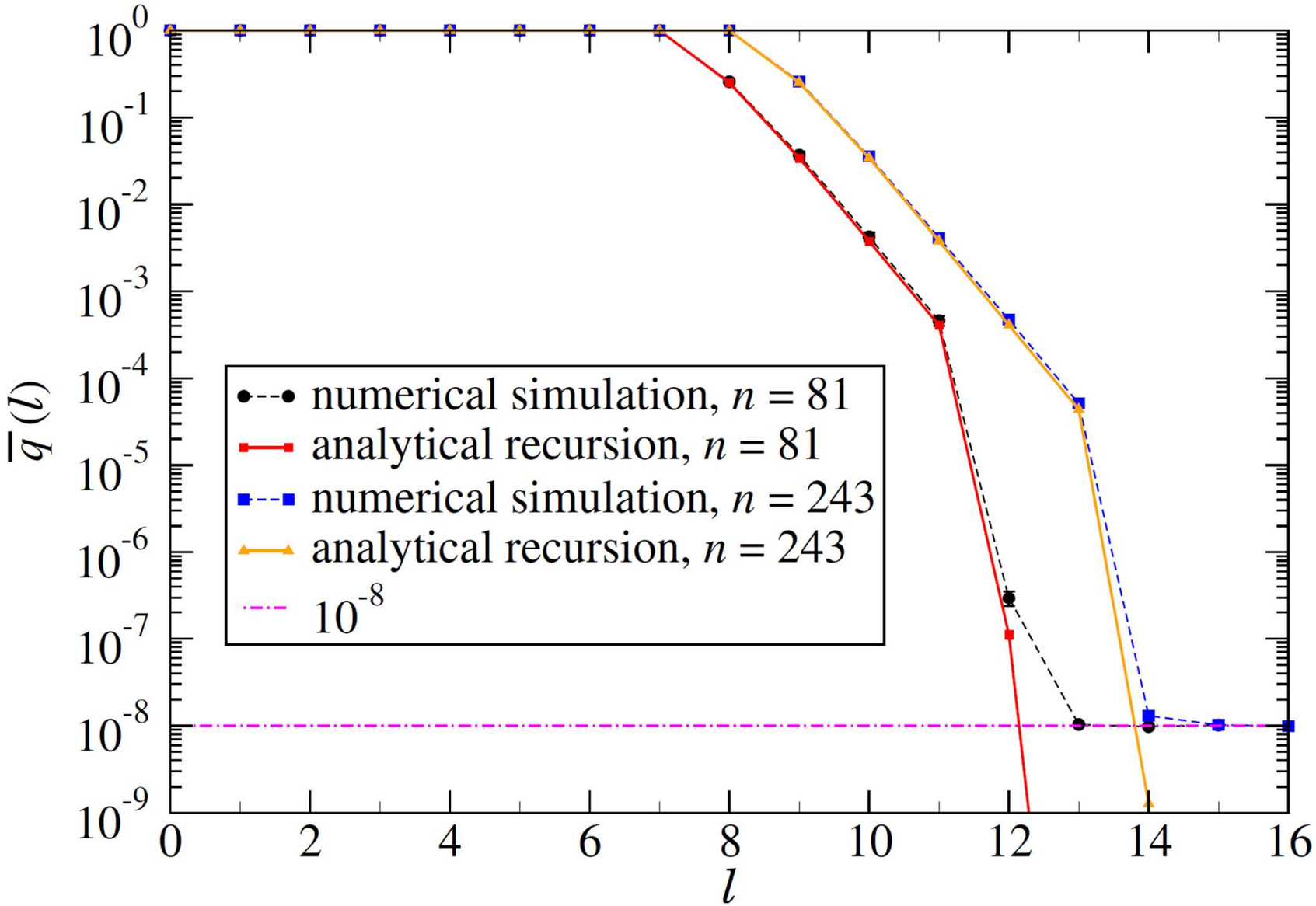
Shows OTOC falling off rapidly with circuit depth.

Numerical results were averaged over a few dozen random instances of the cipher.

- Each tested over 10^8 random input blocks

Numerical results reached a floor of 10^{-8} only because of fixed sample size (number of input blocks) of 10^8.

- Numerical results well validate theoretical predictions below this limit



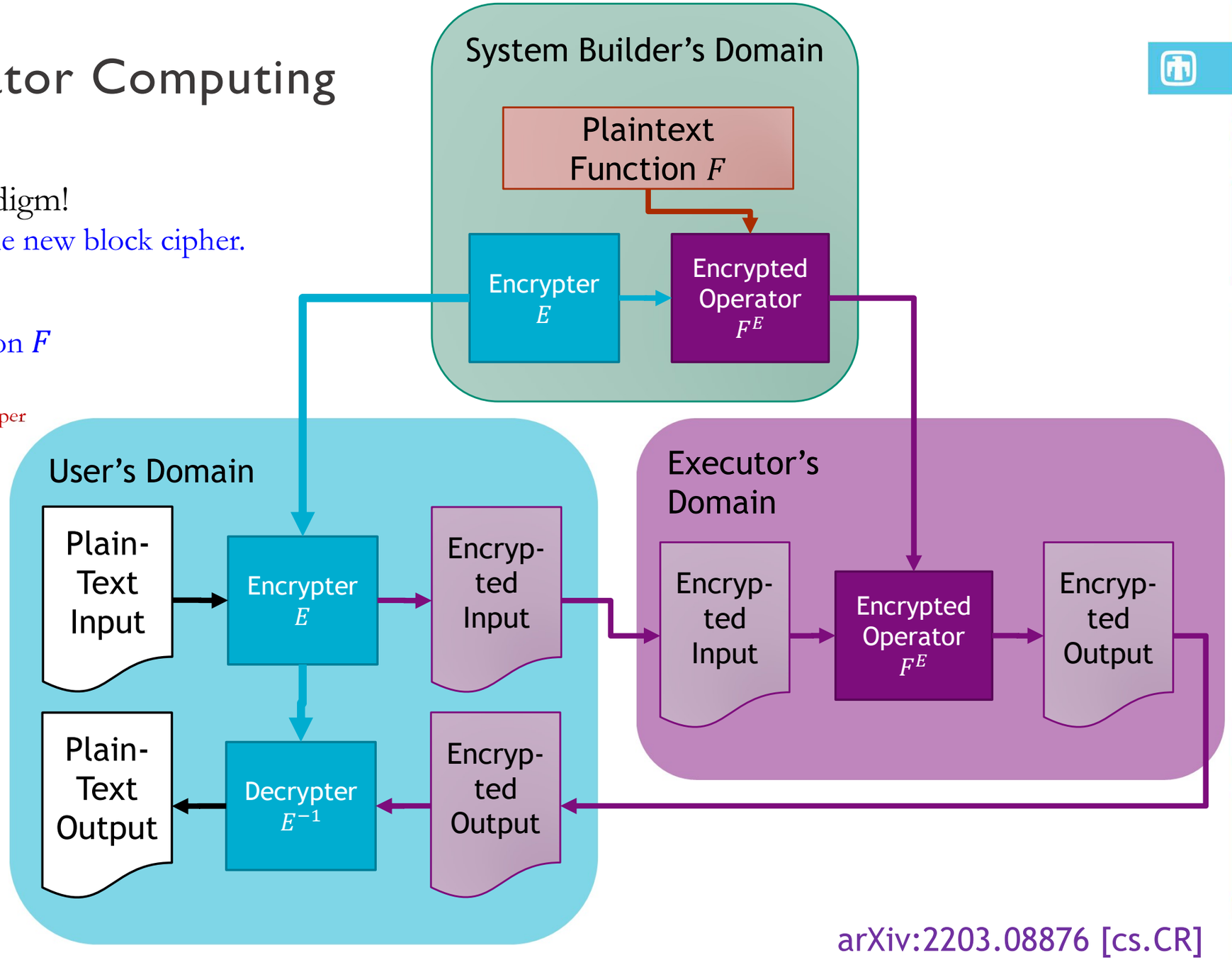
Encrypted Operator Computing (EOC)

A novel secure computing paradigm!

- Based on reversible gates and the new block cipher.

Basic setup:

- The system builder has a function F
  - Generates an encrypter E
    - Pseudo-random cipher as per previous paper
  - E and F are "folded together" to produce an ***encrypted operator*** F^E.
    - **NOTE:** It is infeasible to infer *either* E or F, given just F^E!
  - E and F^E are then given to the ***user*** (input provider / output reader) and ***executor*** (compute server).
    - Can also subdivide E for multiple users with disjoint security domains, for multiparty applications.
- The user has plaintext input
  - Encrypts & sends to executor
- Executor operates in the *encrypted* domain
  - Doesn't need to know *either* the input or the function F!



arXiv:2203.08876 [cs.CR]

Remarks on relationships between EOC & other secure computing paradigms

Comparison with secure Multiparty Computation (MPC) paradigm:

- Like in MPC, in a multiparty setting we can hide users' inputs from each other (& from the executor)
- Like in MPC with garbled universal circuits (as in GABLE), we can also hide the *function being computed* from users (& from the executor), at least in a use-once scenario
- However, the overall overhead factor is larger (polynomial instead of logarithmic).

Comparison with the Fully Homomorphic Encryption (FHE) paradigm:

- In FHE systems, the execution algorithm *does not depend on the specific key* (only on the encryption scheme).
- In contrast, in EOC, the execution algorithm F^E *is derived from the specific key* (the encrypter E), yet *does not reveal the specific key to the executor.*
  - Unlike in FHE, the execution algorithm has to be rebuilt for each different set of users (with different security domains).
- So, it maintains the property that the user data is kept hidden from the entity executing the computation.
  - And, it maintains the property that it can be applied multiple times to different input data from the same set of users.
- Polynomial overheads, can be competitive with FHE

Comparison with the Indistinguishability Obfuscation (IO)/Best-Possible Obfuscation Paradigm:

- Due to the classic Barak impossibility result, if the user and executor domains are *combined together*,
  - the union of them *cannot* completely hide the plaintext function F, however…
- With the domains separated, it does not appear possible for either the user or the executor *by itself* to infer F.
  - Thus, if you can assume *no collusion* between user and executor, we can effectively hide the plaintext function from both.

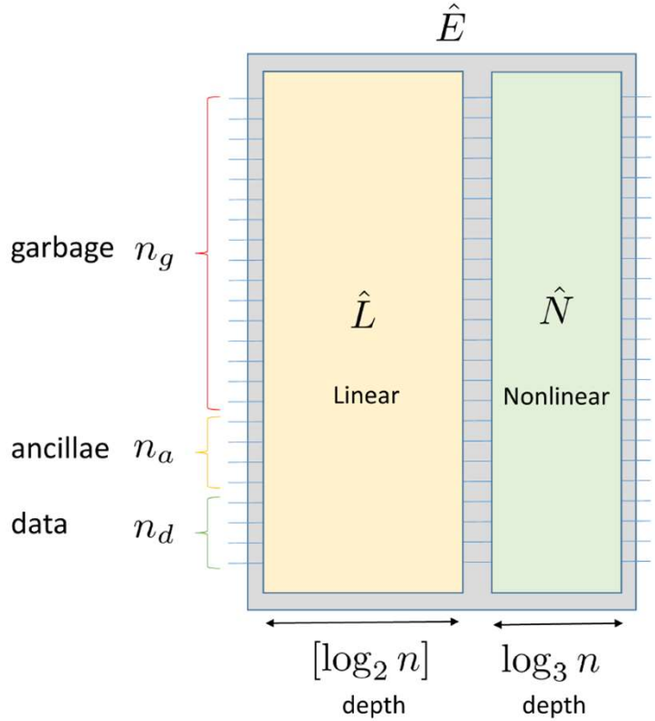
Overview of Methods used in EOC Work

The function to be executed is expressed as a reversible circuit.

- This is always efficiently possible, given some ***ancilla*** bits (temporary memory), and some space for ***garbage*** output bits (generally required for reversibility).

The 3-stage cipher from previous work is able to be simplified to a 2-stage cipher using probabilistic encryption

- *I.e.*, a random salt is added to the plaintext input, so that there are many possible ciphertexts for each plaintext.
- This allows the expression of the encrypter as just $E = NL$,
  - Where L is the linear stage, and N the nonlinear stage.
- This then leads to a two-stage "folding" process, wherein we fold first the linear stage into F (expanding the circuit), and then fold the nonlinear stage into that (further expanding).
  - After each stage, we transform the resulting circuit in ways that obliterate the separation between gates of the original function F and the encryption circuit E.



Further randomization of the resulting circuit takes place by randomly inserting pairs of NOT gates between individual stages ("chips") of the encrypted operator and then absorbing them into the circuits on each side with further transformations.

- There are many more details, but I'm out of time! → Read the paper.

arXiv:2203.08876 [cs.CR]

Conclusion

The Blockchain-Derived Secure Computing LDRD project at Sandia successfully demonstrated a means (GABLE) to achieve functional and data privacy on a blockchain, but it did have some key limitations…

- "Use-once" restriction

The new Encrypted Operator Computing (EOC) method by Chamon *et al.* seems to offer many of the same security properties as GABLE, with some pros/cons:

- **Pro:** Not limited by use-once restriction!
- **Con:** Polynomial instead of logarithmic overhead in implementation.
- **Con:** Security proofs are perhaps not quite completely rigorous yet.

But: In my opinion, the Chamon *et al.* method represents an important fundamental advance in the secure computing field (both a new setup/paradigm, and a new algorithm with some very nice properties), and is well worth further study.

- The authors encourage feedback, and I'd be happy to put anyone in touch with them!

arXiv:2203.08876 [cs.CR]